Secure Directories

David W Chadwick
IS Institute, University of Salford, Salford, M5 4WT, England

Abstract. This paper describes the mechanisms that are needed in order to provide a secure directory service based on the X.500 data model. A brief introduction to the X.500 data model is given followed by an overview of the Lightweight Directory Access Protocol. Security can be provided by three functions: an application level firewall, an authentication mechanism, and an access control scheme. A description of the X.500 and LDAP access control models is presented followed by the authentication methods that have been standardised for LDAPv3. A companion paper describes a directory application firewall.

1. Introduction

The X.500 data model [1,2] specifies a hierarchically structured information tree (called the Directory Information Tree - DIT), where each node (or entry) in the tree comprises a set of attributes. Each attribute comprises an attribute type and one or more attribute values. Each entry in the tree is uniquely identified with a distinguished name, which is composed hierarchically from the relative distinguished name of the entry and of all its superior entries (see Figure 1).

The DIT is typically distributed between potentially many thousands of servers, with each server holding a part of the DIT called a naming context. User requests can be passed between the servers by a process of chaining or referrals [6].

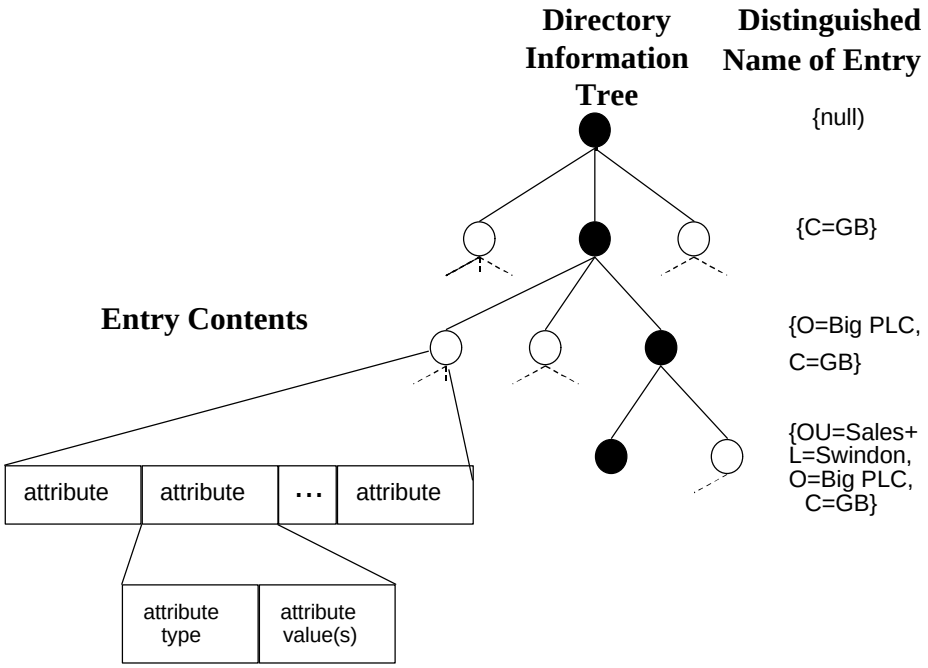

Figure 1 The X.500 Information Model

Directory servers may be accessed by either the DAP [3] or LDAP [4,5] protocols, the latter being far more popular than the former. LDAP is a subset of DAP and comprises the following operations:

Bind - allows the user to log onto the directory server and authenticate to it

Search - allows the user to search and retrieve one or more entries that match some user defined filtering criteria

Compare - allows the user to check if an entry contains a particular attribute value

Abandon - allows the user to abort a Search or Compare operation

Add - allows the user to add a new leaf entry

Modify - allows the user to modify the attributes in an existing entry

Delete - allows the user to delete a leaf entry

ModifyDN - allows the user to change the distinguished name of an entry (and its children) or move an entry to a new point in the DIT

DAP in addition defines Read and List operations that are subsets of the Search operation.

In order to ensure that the information in the directory server is kept secure, i.e. only modified or retrieved by authorised people, various security functions are required. Firstly, we may need a firewall to restrict the protocols arriving at the server from the Internet. A separate paper [8] describes the Guardian DSA, an application proxy firewall that filters the directory protocols DAP, LDAPv2, LDAPv3, DSP (the X.500 chaining protocol) and DISP (the X.500 replication protocol). Secondly the information in the directory may need protecting with access control information to ensure that only authorised people have rights to access the directory data. Section 2 describes the access control scheme defined by X.500 and section 3 describes the access control model for LDAP currently being defined by the IETF. Finally we may need to authenticate the users as they attempt to bind to the directory server, since access controls without effective authentication are meaningless. Section 4 describes the various authentication methods that have been defined for LDAP.

2. The X.500 Access Control Scheme

The X.500 access control scheme was standardised in 1993 [1], and all X.500 based directory server suppliers claim to support it. The general model used for the access control scheme is that a user needs permission to:

a) access the entry

b) access an attribute type

c) access each attribute value.

Access Control Information (ACI) is held as operational attributes within the DIT (operational attributes are not visible to normal directory users but only to administrators). The ACI say which users have what permissions to access which protected data items. The permissions may be to grant access or deny access. Each ACI is given a precedence, with higher precedence ACI over-riding lower precedence ones, and denies over-riding grants of equal precedence, e.g a grant of precedence 20 will override a deny of precedence 10.

Setting permissions can be time consuming and complex. Consequently the X.500 model recognises that administrators will want to set access control policies to control large parts of the directory tree. A Directory Access Control Domain (DACD)

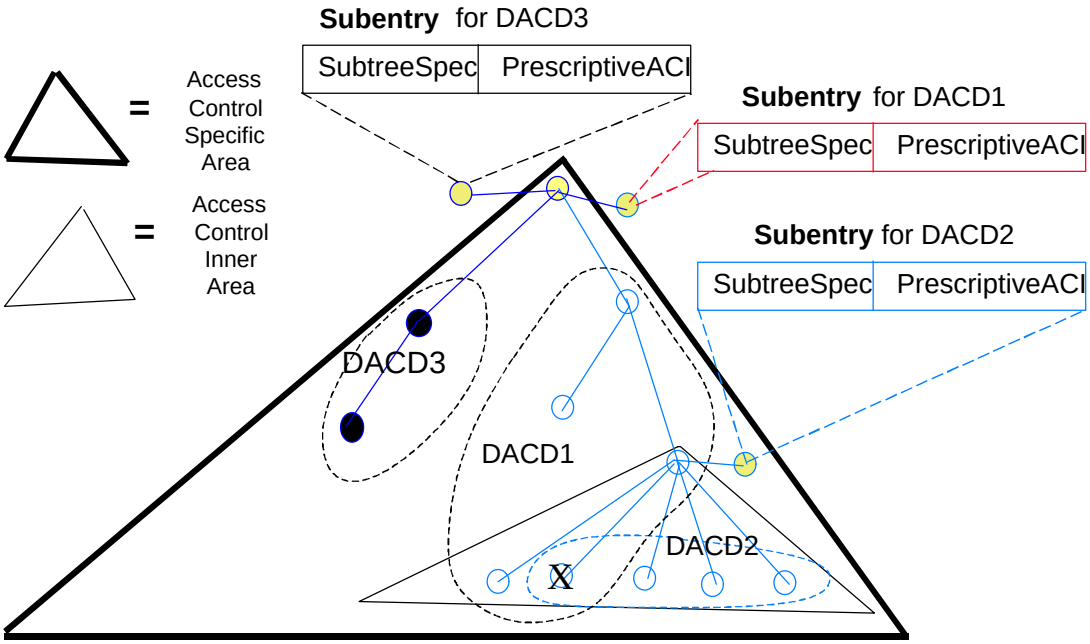

Figure 2 Directory Access Control Domains

is a part of the directory tree that is controlled by an access control policy. Each domain is specified in a special entry called a subentry. Subentries are not visible to normal directory users, but only to administrators. A subentry holds a subtree specification attribute that describes the entries that are in the domain. It also holds a prescriptiveACI attribute that specifies the access control policy for the domain.

The model supports delegation so that we can have access control areas and inner areas. An administrator in charge of an inner area can only create DACDs within the inner area. The administrator in charge of the entire access control area can create DACDs in both the inner and outer areas (see Figure 2).

In the Figure 2 there are three DACDs, specified in the subtree specifications of three subentries. One of the domains, DACD2, is within an access control inner area. Note that DACDs can overlap so that entries may be covered by multiple policies, as is the case for entry X in Figure 2, which is covered by policies DACD1 and DACD2. Also note that whilst DACDs in a specific outer area can cover entries in an inner area, the converse is not true.

Access Control Information can also be tailored to individual entries, and in this case is stored in the EntryACI operational attribute. The syntax/contents of prescriptiveACI and entryACI are the same.

2.1. ACI syntax/content

Access Control Information says which users have what permissions to access which protected data items, providing they have been authenticated to a certain level. The syntax for ACI is specified in ASN.1 [7]. As this is rather complex, this paper will

present the ACI content in textual form. Readers should consult [1] for a complete specification.

The user component of ACI can take the following values:

- **all users** i.e. public access to the entry or entries in question,
- **this entry** is the user whose DN matches the DN of the entry to be accessed,
- **name** is the DN of any object,
- **user group** is the DN of a entry containing a list of names (such as the name of a project team). All names in the list are given the access rights,
- **subtree specification** describes a portion of the directory tree and all users in this part of the tree are granted (or denied) access.

The protected data item can range from the finest granularity of a single attribute value, to a whole domain of entries. The values that protected item can take are:

- one or more **attribute values,**
- **self value** is used for values of distinguished name syntax, and protects the item whose value matches the DN of the accessing user,
- **all attribute values** in a specified attribute
- one or more **attribute types**,
- **all attribute types**
- **all attribute types and values**
- **entry** gives permission to access an entry (but not necessarily its contents), and is either a single entry in the case of entryACI, or a domain of entries in the case of prescriptiveACI.

There is a wide range of permissions that can either be granted or denied. Most permissions are intuitive from their name, and coincide with the operation of the same name, but a few are not. Permissions can be applied to entries or attributes or both. Permissions that apply to all data items are:

- **add** permission allows the protected item to be added
- **remove** permission allows the protected item to be deleted
- **disclose on error** permission allows the user to be given an informative error message about the data item, if the operation fails e.g. “attribute already exists” when trying to add it.

Permissions that only apply to attributes are:

- **compare** allows the attribute to be compared
- **filter match** allows the attribute to be used in a filter of a Search operation. If filter match is not granted, the Search behaves as is this attribute is missing.

Entry level permissions are:

- **browse** allows the entry to be accessed in a Search operation if the user did not specify its DN,
- **export** and **import** are used to move an entry with the ModifyDN operation. Permission must be granted to export from the current location and import to the new location.
- **modify** allows the entry to be modified,
- **rename** allows the entry to be renamed with the ModifyDN operation,
- **return DN** protects the DN of an entry from disclosure. Without this permission, a user will either be given an alias name for an entry, or will not be able to retrieve the entry.

Each ACI has a precedence in the range 0 to 255, zero being the lowest. Higher precedence ACI over-ride lower precedence ACI, and a deny over-ride a grant of equal precedence. For example, an administrator could deny everyone from organisation X access, and grant user Fred from organisation X access with a higher precedence. Alternatively, he could grant all his group access, but deny access to one person from the group. Note that nothing can over-rule a deny of 255 precedence.

For a permission to be granted, a user must be authenticated to at least the level specified in the authentication level of the ACI. If the permission is a deny, then all users who are authenticated below the authentication level will be denied access, as well as those mentioned in the ACI who authenticated at or above the authentication level. This is because a user has to prove, at a high enough level of authentication, that he is not the user who is being denied access.

2.2. Permissions for each operation

The permissions needed for each operation are given in Table 1. One will notice the large number of permissions that are needed for the Search operation. For example to Search a directory tree with a thousand entries using a single filter, approximately three thousand permissions are needed to perform the filtering (entry browse and attribute type and value filter match). It becomes immediately obvious why policy permissions placed on whole domains will substantially improve the performance of Search operations, compared to entry level permissions placed on every entry in the DIT, as in the former case the permissions only need to be evaluated once for the whole domain rather than for every entry.

The observant reader will note an apparent inconsistency between the permissions needed for the Add and Remove operations. The reason for this is that an administrator may wish to delete an entry, but a user with limited update permissions may have added an attribute to his entry giving himself sole permission to delete it. If the remove permissions were similar to the add permissions, then the administrator would not be able to delete the entry using the Remove Entry operation.

Table 1. The Permissions for each Operation

Operation	**Entry Level Permission**	**Attribute Level Permission**
Search	Browse	Filter Match on Attribute Types and Values in Filter Read for types returned Read for values returned
Compare	Read	Compare for type Compare for value
Add Entry	Add	Add for each type and value added
Remove Entry	Remove	none
Modify Entry	Modify	Add for each type and value added Remove for each type and value removed
Modify DN	Rename for change of RDN Export and Import for move	none

3. The LDAP Access Control Scheme

The access control requirements for LDAP accessible directories has been published as an informational RFC [9]. These requirements are sufficiently general to satisfy everyone's needs and so are not particularly contentious. However, defining a standard scheme is proving to be extremely difficult and time consuming. The work started in the IETF LDAPEXT group in 1996. The main problem is that Microsoft, Netscape, Novell, Lotus, X.500 vendors etc. have already implemented their own access control schemes, and getting them to all agree to support another standard LDAP scheme is very difficult (if not impossible). Microsoft for example was against LDAP standardising an access control scheme at all.

The latest model document is "Access Control Model for LDAPv3" [10]. It bears some resemblance to the X.500 access control scheme, and some vendors would like it to be a subset of the X.500 scheme, although whether this will happen or not is doubtful. The LDAP model document is still not fully stable and it is likely to change somewhat before it becomes a proposed standard RFC. It currently specifies:

- how to represent the access control scheme(s) in force at a server
- the LDAP operations for retrieving and updating access control information
- the LDAP attribute for exporting and replicating access control information between servers
- the permissions needed for each LDAP operation
- how access control decisions are made
- how access control information is protected.

However, the document does not specify:

- how access control information is stored by a server as this is implementation dependent.

The access control schemes supported by a server are published by storing the object identifiers of the schemes in an operational attribute in the root entry. Further, the access control scheme actually being applied to each naming context is held in a subentry beneath the root of the naming context. This will allow LDAP clients and servers to see if they are using compatible access control schemes or not.

The model specifies a *ldapACI* operational attribute that may be used to export and replicate LDAP access controls between servers. However the model is clear to indicate that servers need not store their ACI information in this format as it is a local matter how they are stored.

3.1. ACI Syntax/Contents

In keeping with the LDAP philosophy, the syntax of the *ldapACI* operational attribute value is a string rather than an ASN.1 set of octects (as is the case for X.500). The string comprises 4 components separated by the hash (#) character. The attribute can be multi-valued and values can be selected using the caseIgnoreString equality matching rule. The components of an ACI value are:

- Scope. This says if this ACI applies only to the entry it is stored in, or to the subtree beneath the entry.
- Rights consists of grant or deny followed by a set of permissions (see later)

- Attributes is the set of objects the rights apply to. This is either a list of attribute object identifiers, or “all” (meaning all attributes), or “entry” (meaning the entry).
- Subject is who is being controlled by these access control rights, and is an optional authentication level followed by either:
  - a simple **LDAP DN** prefixed by the string “dn:”
  - a **user name** string prefixed by “un:”
  - “**role**:” followed by the DN of a role
  - “**group**:” followed by the DN of a group of names entry, (meaning anyone who is in the list of members has the right)
  - “**subtree**:” followed by the DN of a non-leaf node (meaning anyone whose DN starts with this prefix has the right)
  - “**ipAddress**:” followed by an IP address string (e.g. 1.2.3.4) or a wildcard IP address (e.g. 1.2.3.*) to specify a subnetwork, or a subnetmask (e.g. 1.2.3.*+255.255.255.115) or a wildcard domain name (e.g. *.myorg.com)
  - “**this**:”, meaning the client with the same DN as the entry
  - “**public**:”, meaning anyone.

The optional authentication level says what level of authentication the user needs to have in order to obtain the right. Authentication level is specified as

- any (meaning any type of authentication, or none)
- simple (meaning password)
- “sasl:” followed by “any” or a SASL name [11]

Permissions can be at the entry level or attribute level (as in X.500) but LDAP does not support attribute value level permissions (unlike X.500 which does). LDAP has chosen to give entry and attribute permissions different names (unlike X.500) so that there can be no confusion between the rights being given. Entry level permissions are:

- **add** an entry below this entry
- **delete** the entry (no other permissions needed)
- **export** an entry and all its subordinates from current location
- **import**, allows an entry and all its subordinates to be placed here
- **renameDN**, allows the RDN of an entry to be renamed
- **browseDN**, allows an entry to be accessed in a Search when its DN has not been specified
- **returnDN** allows the DN of the entry to be disclosed in an operation result
- **discloseOnError** says if the authenticated user will get a sensible error message or “entry does not exist”.

Attribute level permissions are:

- **read** allows attribute types and values to be returned in a Search operation
- **write** allows attributes and values to be added in a Modify operation
- **obliterate** allows attributes and values to be deleted in a Modify operation
- **make** allows attributes to be created in an AddEntry operation
- **search** allows attributes and values to be used in a Search filter.
- **compare** allows an attibute and its values to be used in a Compare operation.

3.2. Permissions for each Operation

The permissions needed for each operation are given in Table 2. These are quite similar to those for X.500 (apart from the fact that attribute value level permissions do not exist), but a few noticeable differences occur. For example, the Compare and Modify Entry operations do not require any entry level permissions.

Table 2. The Permissions for each Operation

Operation	**Entry Level Permission**	**Attribute Level Permission**
Search	Browse for each entry in scope ReturnDN for each entry returned	Search on attributes in the Filter Read for attributes returned
Compare	none	Compare for the attribute
Add Entry	Add on parent of entry	Make for each attribute added
Remove Entry	Delete	none
Modify Entry	none	Write for each attribute added Obliterate for each attribute removed
Modify DN	RenameDN for change of RDN Export and Import for move	none

In conclusion, the LDAP access control model is simpler than the X.500 access control model, but is not a true subset of it. Therefore X.500 based LDAP servers would need to make some changes to their products in order to comply with the LDAP model as currently specified, although it must be noted that the model is still not fully stable. One hopes that it might be published as an RFC in 2001.

4. The Authentication Methods for LDAP

LDAPv2 [4] documents the use of anonymous binds, clear password based authentication and Kerberos authentication. However, as LDAPv2 has been superseded by LDAPv3, these methods will not be progressed further. The authentication methods for LDAPv3 [12] are currently at proposed Internet standard status, and will be progressed to full standard status in due course. [12] documents which authentication methods are mandatory and which are optional to be supported by every LDAPv3 server. In keeping with the IESG policy, clear passwords are not required to be supported, and are in fact discouraged from being used. The mandatory methods of authentication that each LDAP server must support are: anonymous Binds (for public access) and Digest-MD5 Binds [13] which use hashed passwords. The optional authentication methods that should be used if confidential communications are required are: simple password authentication over an encrypted TLS session [14], or X.509 certificate based authentication with TLS.

LDAPDigest-MD5 authentication is compatible with HTTP/1.1 Digest Authentication “md5-sess” [15]. It is a two phase process and works as follows. The client sends a Bind Request with the SASL mechanism set to “DIGEST-MD5” and the server sends a Bind Response containing a digest challenge and a result code of saslBindInProgress. The client sends a second Bind Request containing the digest response and the server sends a Bind Response with a success or error code. The digest challenge contains various parameters including the name of the server’s realm, a random nonce, a quality of protection parameter (authentication, authentication + integrity or authentication + confidentiality), a maximum buffer size, a character set of

UTF-8, and a choice of algorithms for encryption (DES, triple-DES, or RC4). The digest response contains a hash of the user's name, the realm, the user's password, the original nonce and a client generated nonce. The reply may optionally be encrypted.

Transport Layer Security (TLS) is the Internet standard and enhanced version of SSLv3 [16] (although the mandatory to implement algorithms are not compatible between the two protocols). The use of TLS by LDAP can be found in [17]. An LDAPv3 extended operation "Start TLS" is defined, which can be sent at any time between the client and server, providing of course that there are no outstanding LDAP responses, or a SASL Bind is not already in progress. If the server supports TLS sessions it will reply OK and the TLS negotiation will then start. After the TLS encrypted link has been established, if simple password authentication is being used, the client sends a Bind Request containing its name and password. If however mutual X.509 certificate based authentication was used by the TLS implementation, then the client sends a Bind Request with the authentication method set to SASL EXTERNAL, and the server has to obtain the clients authenticated identity from the TLS implementation.

LDAPv3 currently does not document how alternative authentication mechanisms might be used, such as Kerberos or S-Key.

5. Conclusion

Directory servers may hold vital organisation information, and if so it may be essential that this information is only made available to authorised users. This paper has described the X.500 and LDAP based access control models for allocating access rights to the information to users, and has noted some similarities and differences between the two models. However, without proper user authentication, any access control scheme has little value if users can easily masquerade as others. Since the most popular way of accessing X.500 based directories is with LDAP, the paper then describes the authentication methods that are being standardised for LDAPv3. These methods range from the popular though insecure anonymous access to the highly secure use of X.509 certificates and digital signatures over an encrypted TLS link.

References

[1] ISO/ITU-T Recommendation X.501: "The Directory - Models". 1993.
[2] D.W.Chadwick. "Understanding X.500", Chapman and Hall, 1994. Also available from http://www.salford.ac.uk/its024/X500.htm
[3] ISO/ITU-T Recommendation X.511: "The Directory - Abstract Service Definition". 1993.
[4] Yeong, W., Howes, T., and Kille, S. "Lightweight Directory Access Protocol", RFC 1777, March 1995.
[5] M. Wahl, T. Howes, S. Kille, "Lightweight Directory Access Protocol (v3)", Dec. 1997, RFC 2251
[6] ISO/ITU-T Rec. X.518(1993) The Directory: Procedures for Distributed Operation
[7] ITU-T Rec. X.680 ¦ ISO/IEC 8824-1. Abstract Syntax Notation One (ASN.1): Specification of basic notation
[8] D.W.Chadwick, A.J. Young. "A Directory Application Firewall - the Directory Guardian", companion paper to this one
[9] E. Stokes, D. Byrne, B. Blakley, P. Behera. "Access Control Requirements for LDAP". RFC 2820, May 2000.

[10] E. Stokes, D. Byrne, B. Blakley. “Access Control Model for LDAPv3” <draft-ietf-ldapext-acl-model-06.txt>, 14 July 2000.
[11] J.Myers. “Simple Authentication and Security Layer (SASL)”. RFC 2222, October 1997
[12] Wahl, M., Alverstrand, H., Hodges, J., Morgan, R. “Authentication Methods for LDAP”, RFC 2829, May 2000
[13] Leach, P., Newman, C. “Using Digest Authentication as a SASL mechanism”, RFC 2831, May 2000.
[14] Dierks, T., Allen, C. "The TLS Protocol Version 1.0", RFC 2246, January 1999.
[15] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., Stewart, L. “HTTP Authentication: Basic and Digest Access Authentication”. RFC 2617, June 1999.
[16] Frier, A., Karlton, P., Kocher, P. ‘The SSL 3.0 Protocol’, Netscape Communications Corp., Nov 1996
[17] Hodges, J., Morgan, R., Wahl, M. “Lightweight Directory Access Protocol (v3): Extension for Transport Layer Security”. RFC 2830, May 2000.